# EXHIBIT 3

# U.S. Patent No. 7,519,814 vs. HPE

Accused Instrumentalities: HPE products and services using secure containerized applications, including without limitation HPE's Ezmeral Runtime Enterprise and HPE GreenLake, and all versions and variations thereof since the issuance of the asserted patent.

Each Accused Instrumentality infringes the claims in substantially the same way, and the evidence shown in this chart is similarly applicable to each Accused Instrumentality. Each claim limitation is literally infringed by each Accused Instrumentality. However, to the extent any claim limitation is not met literally, it is nonetheless met under the doctrine of equivalents because the differences between the claim limitation and each Accused Instrumentality would be insubstantial, and each Accused Instrumentality performs substantially the same function, in substantially the same way, to achieve the same result as the claimed invention. Notably, Defendant has not yet articulated which, if any, particular claim limitations it believes are not met by the Accused Instrumentalities.

## Claim 1

| Claim 1 | Accused Instrumentalities |
|---|---|
| [1pre] 1. In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising: | To the extent the preamble is limiting, HPE and/or its customer practices, through the Accused Instrumentalities, in a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, as claimed.<br><br>For example, HPE Ezmeral Runtime Enterprise runs on individual servers, including HPE Synergy and HPE ProLiant servers, each of which runs an independent operating system, including for example RHEL or SLES running either on bare metal, through an on-premises virtualized infrastructure, through one or more cloud services, or through any other supported deployment. In an exemplary deployment, two or more servers use different operating systems.<br><br>HPE requires that each server includes a processor with one or more cores available to the OS kernel. HPE further requires each server to have a supported operating system (SLES or RHEL/CentOS), which includes a kernel and associated local system files, including for example libraries such as libc/glibc, configuration files, etc. In the infringing system, at least two servers have different |

| Claim 1 | Accused Instrumentalities |
|---|---|
| | operating systems, for example SLES and RHEL/CentOS, or for another example different versions of SLES and/or RHEL/CentOS.<br><br>In at least some instances, HPE directly owns, operates, controls, and/or benefits from the claimed system and/or method. In other instances, HPE's customer makes and uses the system and/or method either by following HPE's direction and control, including HPE's documentation, or automatically through the ordinary and expected operation of HPE's software, or a combination thereof.<br><br>*See* claim limitations below.<br><br>*See also, e.g.*:<br>HPE Ezmeral Runtime Enterprise is an enterprise-grade container orchestration platform that is designed for the containerization of both cloud-native and non-cloud-native monolithic applications with persistent data. It deploys 100% open-source Kubernetes for orchestration, provides a state-of-the-art file system and data fabric for persistent container storage, and provides enterprises with the ability to deploy non-cloud-native AI and Analytics workloads in containers. Enterprises can now easily extend the agility and efficiency benefits of containers to more of their enterprise applications—running on either bare-metal or virtualized infrastructure, on-premises, in multiple clouds, or at the edge.<br>https://www.hpe.com/psnow/doc/a50004264enw.pdf?jumpid=in_pdp-psnow-qs<br><br>The offering formerly known as the HPE Ezmeral Container Platform is really focused on a lot more than just containers, and it provides businesses with more than just container orchestration software. The name change to HPE Ezmeral Runtime Enterprise reflects the fact that this is not just a solution for container platform orchestration. This platform offers an incredible wealth of capabilities and features you can use to modernize, deploy, monitor, and manage your applications.<br>https://community.hpe.com/t5/hpe-ezmeral-uncut/hpe-ezmeral-container-platform-is-now-hpe-ezmeral-runtime/ba-p/7151720<br><br>- **OS agnostic** – With an application and all its necessary files bundled into one unit – minus an operating system – the container can run on different operating systems, hardware, networks, storage systems and security policies. This means that any environment is compatible, so developers don't need to re-write applications for different servers.<br>https://www.hpe.com/us/en/what-is/caas.html |

| Claim 1 | Accused Instrumentalities |
| --- | --- |
| | |

With HPE Ezmeral Runtime Enterprise running on HPE Synergy or HPE ProLiant servers, enterprises can extend the agility and efficiency benefits of containers to more of their enterprise applications—running on either bare metal or virtualized infrastructure, either on-premises, in multiple public clouds, or at the edge.

https://www.hpe.com/psnow/doc/a50003599enw

**HPE Ezmeral Runtime Enterprise and HPE Ezmeral ML Ops Capabilities Matrix**

| | HPE Ezmeral Runtime Enterprise Essentials | HPE Ezmeral Runtime Enterprise | HPE Ezmeral MLOps |
| --- | --- | --- | --- |
| **Operating Systems (OS)** | Yes | Yes | Yes |
| RHEL OS | Yes | Yes | Yes |
| SLES OS | Yes | Yes | Yes |

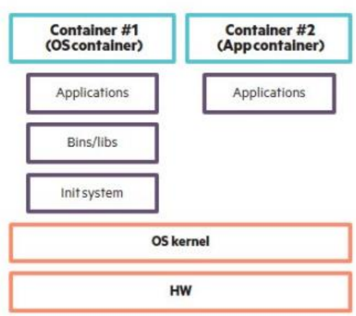
https://www.hpe.com/psnow/doc/a50004264enw.pdf?jumpid=in_pdp-psnow-qs

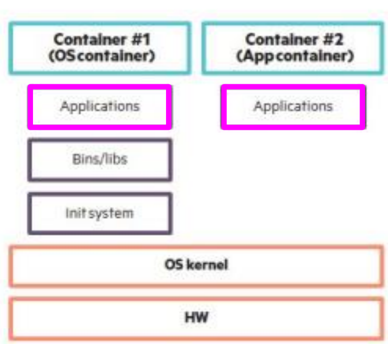
## Standard Features

- Leverages portability of containers to run on any infrastructure (HPE or non-HPE) and any public cloud

https://www.hpe.com/psnow/doc/a50004264enw.pdf?jumpid=in_pdp-psnow-qs

**Two Linux containers on a single system**



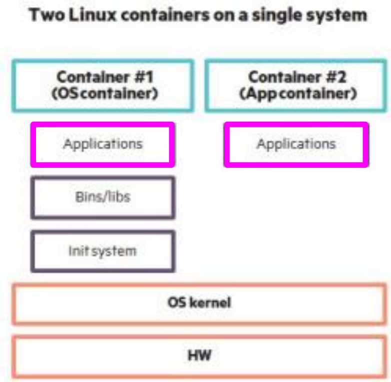
| Claim 1 | Accused Instrumentalities |
|---|---|
| | https://h50146.www5.hpe.com/products/software/oe/linux/mainstream/support/whitepaper/pdfs/4AA6-2761ENW.pdf<br><br>Each license allows the customer to deploy the HPE Ezmeral Container Platform on one Core and 2 terabytes of Storage Capacity. The customer must purchase more licenses if they exceed the allowable amount of Cores or Storage Capacity. As used in this Agreement, Core means a part of a CPU that executes a single stream of compiled instruction code. Each physical processor contains smaller processing units called physical CPU cores. Some processors have two cores, some four, some eight, and so on. Core capacity represents the total number of cores available within a given system. The number of cores is counted as the number of logical cores presented to the product guest OS. For licensing purposes, the number of cores on a given Ezmeral Container Platform host is the number of unique cores available to the kernel in the OS on which the Ezmeral Container Platform software is directly installed, regardless of the number of threads in each core. It equals the product of Core(s) per socket and Socket(s), as shown in the output of<br>https://docs.ezmeral.hpe.com/runtime-enterprise/56/home/about-hpe-ezmeral-container-pl/GEN_End_User_Software_Agreement.html<br><br>Kernel mode refers to the processor mode that enables software to have full and unrestricted access to the system and its resources. The OS kernel and kernel drivers, such as the file system driver, are loaded into protected memory space and operate in this highly privileged kernel mode.<br>https://www.techtarget.com/searchdatacenter/definition/kernel<br><br>Instead of using a hypervisor to manage VMs, the figure shows how containers isolate applications into separate environments (containers) that include processor, memory, and networking resources as part of the container itself. This environment provides OS-level virtualization. Containers have their own root; and, users and processes do not perform operations outside of the container environment. The host OS kernel manages container workloads directly, which reduces the overhead involved with managing system resources. This improves efficiency and therefore, improves performance.<br>**Two Linux containers on a single system**<br> |

| Claim 1 | Accused Instrumentalities |
| --- | --- |
| | https://h50146.www5.hpe.com/products/software/oe/linux/mainstream/support/whitepaper/pdfs/4AA6-2761ENW.pdf<br><br>**Docker container:** A *Docker container* is a lightweight, standalone, executable software package that runs specific services. This software package includes code, runtime, system libraries, configurations, etc. that run as an isolated process in user space. A Docker container is typically used to deploy scalable and repeatable *microservices*. HPE Ezmeral Runtime Enterprise contains innovations around storage, networking, and security to utilize Docker containers as lightweight virtual machines to run Big Data and analytics applications.<br><br>https://support.hpe.com/hpesc/public/docDisplay?docId=a00ecp54hen_us&docLocale=en_US&page=home/about-hpe-ezmeral-container-pl/GEN_Definitions.html<br><br>Controller, Gateway, and Worker Hosts<br><br>A host is either a physical server or a virtual server, located on your premises or in a public cloud, that is available to HPE Ezmeral Runtime Enterprise. The term **host** and **node** are often used interchangeably. Nodes are hosts that are part of a cluster.<br><br>You must have a supported operating system installed on hosts before they can be used in HPE Ezmeral Runtime Enterprise. Hosts have different requirements depending on their functions. See Host Requirements.<br><br>https://docs.ezmeral.hpe.com/runtime-enterprise/56/reference/universal-concepts/Controller_Gateway_and_Worker_Hosts.html |

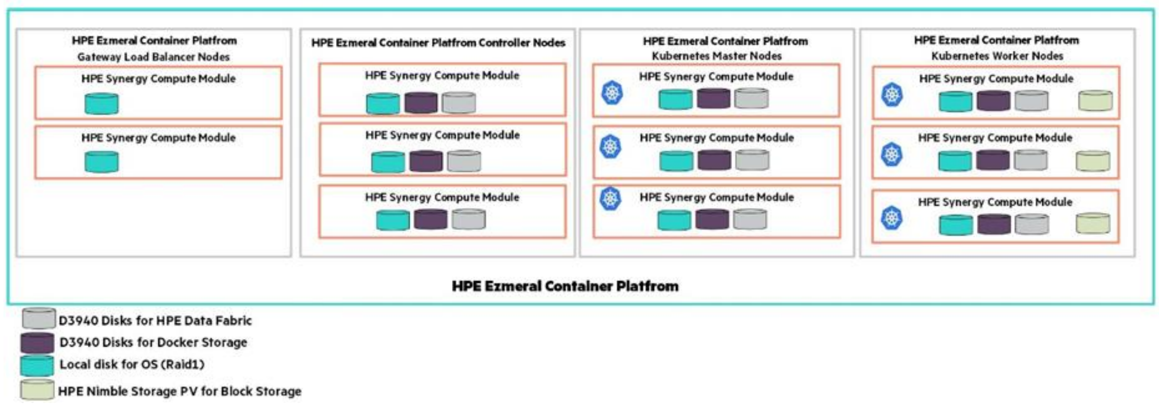
| Claim 1 | Accused Instrumentalities |
| --- | --- |
| | **Kubernetes Cluster Nodes**<br><br>A deployment of HPE Ezmeral Runtime Enterprise can include multiple Kubernetes clusters. A host that is part of a Kubernetes cluster is referred to in Kubernetes as a node.<br><br>Each Kubernetes cluster has its own control plane, consisting of at least one control plane node. The Kubernetes control plane is separate from the Platform Control Plane. A high-availability Kubernetes cluster has multple control plane nodes, as described in High Availability.<br><br>Kubernetes clusters contain worker nodes that run the containers and pods that process jobs in HPE Ezmeral Runtime Enterprise.<br><br>For more information about hosts and Kubernetes clusters, see Controller, Gateway, and Worker Hosts.<br><br>https://docs.ezmeral.hpe.com/runtime-enterprise/56/reference/kubernetes/Kubernetes_Physical_Architecture.html#v52_k8s-kubernetes-physical-architecture__k8s-cluster-architecture |
| [1a] storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers; | The method practiced by HPE and/or its customer through the Accused Instrumentalities includes a step of storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers.<br><br>For example, HPE Ezmeral Runtime Enterprise stores application containers, sometimes called Docker containers, container images, Kubernetes containers, or Kubernetes pods, in persistent storage available to each node running the application. The container might be in a format defined by the Open Container Initiative. This storage may be physically attached to the server or connected through any supported interconnect, including over a network. Each container includes the application software as well as a Linux user space required to execute the application, for example libc/glibc and other shared libraries, configuration files, etc. necessary for the application. For example, the container includes a base OS image, provided by HPE or by a third party, such as a CentOS, RHEL, or Ubuntu base image. The container is compatible with the host kernel, for example because the |

| Claim 1 | Accused Instrumentalities |
|---|---|
| | container libraries are linked against the Linux kernel, and the supported host operating systems also use the Linux kernel, which has a stable binary interface.<br><br>*See, e.g.*:<br><br>**Pod:** For Kubernetes, a *pod* is a group of containers deployed on a single host.<br><br>**Data Fabric cluster:** This is a Kubernetes cluster that is used for HPE Ezmeral Data Fabric storage. A Data Fabric cluster is a Custom Resource in Kubernetes that is supported by operators in HPE Ezmeral Runtime Enterprise.<br><br>**Data Fabric CR:** This typically refers to the Custom Resource specification for a Data Fabric cluster that is supported by an HPE Ezmeral Runtime Enterprise `dataplatform` operator. It specifies each type of pod that the cluster would comprise. The per-pod specification may include CPU, memory, disk, and port requirements. Together with node labels and annotations, the Data Fabric CR influences the placement and scheduling of cluster pods by Kubernetes. HPE Ezmeral Runtime Enterprise creates and applies the Data Fabric CR when creating the first Data Fabric cluster. The Data Fabric CR may be subsequently patched/modified when expanding the cluster, or by a user with suitable privileges.<br><br>**Docker container:** A *Docker container* is a lightweight, standalone, executable software package that runs specific services. This software package includes code, runtime, system libraries, configurations, etc. that run as an isolated process in user space. A Docker container is typically used to deploy scalable and repeatable *microservices*. HPE Ezmeral Runtime Enterprise contains innovations around storage, networking, and security to utilize Docker containers as lightweight virtual machines to run Big Data and analytics applications.<br><br>https://support.hpe.com/hpesc/public/docDisplay?docId=a00ecp54hen_us&docLocale=en_US&page=home/about-hpe-ezmeral-container-pl/GEN_Definitions.html |

| Claim 1 | Accused Instrumentalities |
|---|---|




https://h50146.www5.hpe.com/products/software/oe/linux/mainstream/support/whitepaper/pdfs/4AA6-2761ENW.pdf




**Storage**

The HPE Synergy D3940 Storage Module provides solid state disks and hard disk drives to local systems where it is consumed by HPE Ezmeral Data Fabric, and optionally by compute nodes as boot devices. HPE Nimble Storage dynamically provides Persistent Volume (PV) for containers using Dynamic Volume Provisioner which is integrated with the HPE CSI Driver.

https://www.hpe.com/psnow/doc/a50002075enw

**HPE Nimble Storage**

HPE Nimble Storage AF40 is used to provide persistent, block storage in this solution. The HPE Nimble Storage array for Docker data provides the storage volume to host the repository, to store container images, and also provides persistent volume for applications.

| Claim 1 | Accused Instrumentalities |
|---|---|
| | https://www.hpe.com/psnow/doc/a50002075enw<br><br>## Container images<br><br>A container image is a ready-to-run software package containing everything needed to run an application: the code and any runtime it requires, application and system libraries, and default values for any essential settings.<br><br>https://kubernetes.io/docs/concepts/containers/<br><br>An application container is a stand-alone, all-in-one package for a software application. Containers include the application binaries, plus the software dependencies and the hardware requirements needed to run, all wrapped up into an independent, self-contained unit.<br><br>https://developer.hpe.com/blog/kubernetes-application-containers-managing-containers-and-cluster-resour/<br><br>Because each application container creates an isolated environment for its application, the resources allocated to it are the entire machine. Other copies of the same container are "unaware" of each other.<br><br>https://developer.hpe.com/blog/kubernetes-application-containers-managing-containers-and-cluster-resour/<br><br>- **OS agnostic** – With an application and all its necessary files bundled into one unit – minus an operating system – the container can run on different operating systems, hardware, networks, storage systems and security policies. This means that any environment is compatible, so developers don't need to re-write applications for different servers.<br><br>https://www.hpe.com/us/en/what-is/caas.html<br><br>6. **Do Docker containers package up the entire OS and make it easier to deploy?**<br><br>Docker containers do not package up the OS. They package up the applications with everything that the application needs to run. The engine is installed on top of the OS running on a host. Containers share the OS kernel allowing a single host to run multiple containers.<br><br>https://www.docker.com/blog/the-10-most-common-questions-it-admins-ask-about-docker/ |

| Claim 1 | Accused Instrumentalities |
|---|---|
| | Kubernetes namespaces have the following uses:<br>• **Isolation:** Teams, projects, and customers exist in their own environment within a cluster, and do not impact each other's work.<br>https://support.hpe.com/hpesc/public/docDisplay?docId=a00ecp55hen_us&docLocale=en_US&page=reference/universal-concepts/Namespaces.html<br><br>Using containers isolates software and allows it to work independently across different operating systems, hardware, networks, storage systems, and security policies. It allows the container-based application to transition seamlessly through development, testing, and production environments. Because an operating system is not packed into the container, each container uses minimal computing resources, making it light and easy to install.<br>https://www.hpe.com/us/en/what-is/containers.html |

| Claim 1 | Accused Instrumentalities |
|---|---|
| | # About storage drivers<br><br>To use storage drivers effectively, it's important to know how Docker builds and stores images, and how these images are used by containers. You can use this information to make informed choices about the best way to persist data from your applications and avoid performance problems along the way.<br><br>## Storage drivers versus Docker volumes<br><br>Docker uses storage drivers to store image layers, and to store data in the writable layer of a container. The container's writable layer doesn't persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime. Storage drivers are optimized for space efficiency, but (depending on the storage driver) write speeds are lower than native file system performance, especially for storage drivers that use a copy-on-write filesystem. Write-intensive applications, such as database storage, are impacted by a performance overhead, particularly if pre-existing data exists in the read-only layer.<br><br>Use Docker volumes for write-intensive data, data that must persist beyond the container's lifespan, and data that must be shared between containers. Refer to the volumes section to learn how to use volumes to persist data and improve performance.<br><br>https://docs.docker.com/storage/storagedriver/ |

<table>
<tr><th>Claim 1</th><th>Accused Instrumentalities</th></tr>
<tr><td></td><td>

## Images and layers

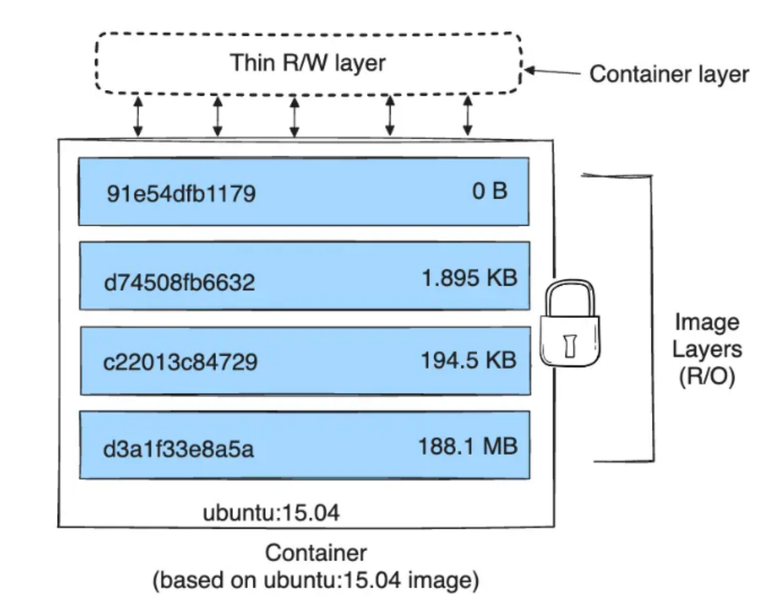
A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only. Consider the following Dockerfile:

```
# syntax=docker/dockerfile:1

FROM ubuntu:22.04
LABEL org.opencontainers.image.authors="org@example.com"
COPY . /app
RUN make /app
RUN rm -r $HOME/.cache
CMD python /app/app.py
```

This Dockerfile contains four commands. Commands that modify the filesystem create a layer. The `FROM` statement starts out by creating a layer from the `ubuntu:22.04` image. The `LABEL` command only modifies the image's metadata, and doesn't produce a new layer. The `COPY` command adds some files from your Docker client's current directory. The first `RUN` command builds your application using the `make` command, and writes the result to a new layer. The second `RUN` command removes a cache directory, and writes the result to a new layer. Finally, the `CMD` instruction specifies what command to run within the container, which only modifies the image's metadata, which doesn't produce an image layer.

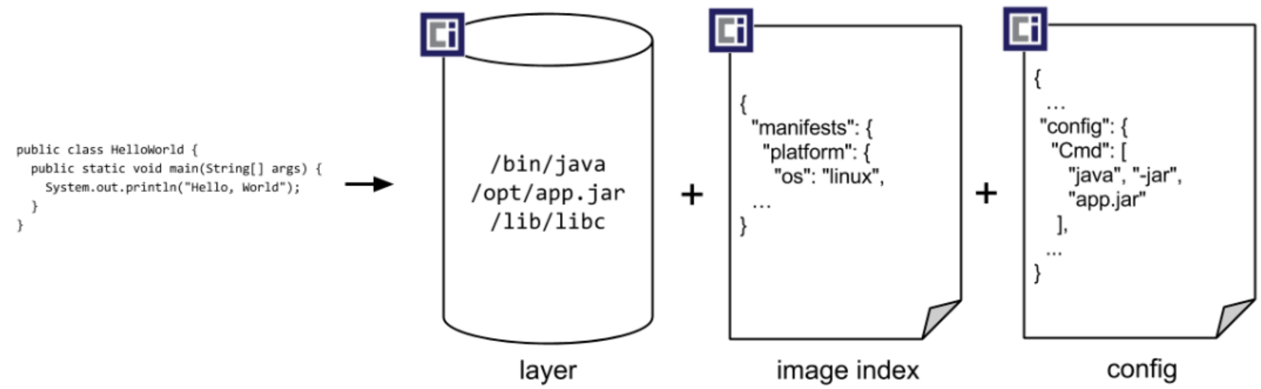
https://docs.docker.com/storage/storagedriver/
</td></tr>
</table>

| Claim 1 | Accused Instrumentalities |
| --- | --- |
| | Each layer is only a set of differences from the layer before it. Note that both *adding*, and *removing* files will result in a new layer. In the example above, the `$HOME/.cache` directory is removed, but will still be available in the previous layer and add up to the image's total size. Refer to the Best practices for writing Dockerfiles and use multi-stage builds sections to learn how to optimize your Dockerfiles for efficient images.<br><br>The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an `ubuntu:15.04` image.<br><br><br>https://docs.docker.com/storage/storagedriver/ |

| Claim 1 | Accused Instrumentalities |
|---|---|
| | # Volumes<br><br>Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While bind mounts are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker. Volumes have several advantages over bind mounts:<br><br>https://kubernetes.io/docs/concepts/storage/volumes/<br><br>## Container environment<br><br>The Kubernetes Container environment provides several important resources to Containers:<br><br>• A filesystem, which is a combination of an image and one or more volumes.<br>• Information about the Container itself.<br>• Information about other objects in the cluster.<br><br>https://kubernetes.io/docs/concepts/containers/container-environment/ |

| Claim 1 | Accused Instrumentalities |
|---|---|
| | **Images**<br><br>A container image represents binary data that encapsulates an application and all its software dependencies. Container images are executable software bundles that can run standalone and that make very well defined assumptions about their runtime environment.<br><br>You typically create a container image of your application and push it to a registry before referring to it in a Pod.<br><br>https://kubernetes.io/docs/concepts/containers/images/<br><br>**Volumes**<br><br>On-disk files in a container are ephemeral, which presents some problems for non-trivial applications when running in containers. One problem occurs when a container crashes or is stopped. Container state is not saved so all of the files that were created or modified during the lifetime of the container are lost. During a crash, kubelet restarts the container with a clean state. Another problem occurs when multiple containers are running in a `Pod` and need to share files. It can be challenging to setup and access a shared filesystem across all of the containers. The Kubernetes volume abstraction solves both of these problems. Familiarity with Pods is suggested.<br><br>https://kubernetes.io/docs/concepts/storage/volumes/ |

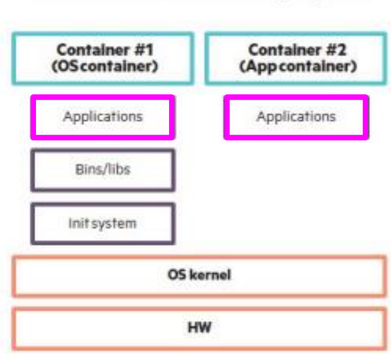
| Claim 1 | Accused Instrumentalities |
|---|---|
| | **Open Container Initiative**<br><br>**Image Format Specification**<br><br>This specification defines an OCI Image, consisting of an image manifest, an image index (optional), a set of filesystem layers, and a configuration.<br><br>The goal of this specification is to enable the creation of interoperable tools for building, transporting, and preparing a container image to run.<br><br>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md |

| Claim 1 | Accused Instrumentalities |
|---|---|
| | ## Overview<br><br>At a high level the image manifest contains metadata about the contents and dependencies of the image including the content-addressable identity of one or more filesystem layer changeset archives that will be unpacked to make up the final runnable filesystem. The image configuration includes information such as application arguments, environments, etc. The image index is a higher-level manifest which points to a list of manifests and descriptors. Typically, these manifests may provide different implementations of the image, possibly varying by platform or other attributes.<br><br><br><br>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md |

| Claim 1 | Accused Instrumentalities |
|---|---|
| | # OCI Image Configuration<br><br>An OCI *Image* is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. This specification outlines the JSON format describing images for use with a container runtime and execution tool and its relationship to filesystem changesets, described in Layers.<br><br>This section defines the `application/vnd.oci.image.config.v1+json` media type.<br><br>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md |

| Claim 1 | Accused Instrumentalities |
|---|---|
| | **Layer**<br><br>• Image filesystems are composed of *layers*.<br>• Each layer represents a set of filesystem changes in a tar-based layer format, recording files to be added, changed, or deleted relative to its parent layer.<br>• Layers do not have configuration metadata such as environment variables or default arguments - these are properties of the image as a whole rather than any particular layer.<br>• Using a layer-based or union filesystem such as AUFS, or by computing the diff from filesystem snapshots, the filesystem changeset can be used to present a series of image layers as if they were one cohesive filesystem.<br><br>**Image JSON**<br><br>• Each image has an associated JSON structure which describes some basic information about the image such as date created, author, as well as execution/runtime configuration like its entrypoint, default arguments, networking, and volumes.<br>• The JSON structure also references a cryptographic hash of each layer used by the image, and provides history information for those layers.<br>• This JSON is considered to be immutable, because changing it would change the computed ImageID.<br>• Changing it means creating a new derived image, instead of changing the existing image.<br><br>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md |

| Claim 1 | Accused Instrumentalities |
| --- | --- |
| | • **rootfs** *object*, REQUIRED<br>The rootfs key references the layer content addresses used by the image. This makes the image config hash depend on the filesystem hash.<br>○ **type** *string*, REQUIRED<br>MUST be set to `layers`. Implementations MUST generate an error if they encounter a unknown value while verifying or unpacking an image.<br>○ **diff_ids** *array of strings*, REQUIRED<br>An array of layer content hashes (`DiffIDs`), in order from first to last.<br>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md |
| [1b] wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems, | In the method practiced by HPE and/or its customer through the Accused Instrumentalities, the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems.<br>The system files in the container are compatible with the host kernel, for example because they are linked against the Linux kernel and the supported host operating systems also use the Linux kernel, which has a stable binary interface.<br>*See* discussion in element [1a] above.<br>*See, e.g.*:<br>**Docker container:** A *Docker container* is a lightweight, standalone, executable software package that runs specific services. This software package includes code, runtime, system libraries, configurations, etc. that run as an isolated process in user space. A Docker container is typically used to deploy scalable and repeatable *microservices*. HPE Ezmeral Runtime Enterprise contains innovations around storage, networking, and security to utilize Docker containers as lightweight virtual machines to run Big Data and analytics applications. |

| Claim 1 | Accused Instrumentalities |
| --- | --- |
| | https://support.hpe.com/hpesc/public/docDisplay?docId=a00ecp54hen_us&docLocale=en_US&page=home/about-hpe-ezmeral-container-pl/GEN_Definitions.html<br><br>• **OS agnostic** – With an application and all its necessary files bundled into one unit – minus an operating system – the container can run on different operating systems, hardware, networks, storage systems and security policies. This means that any environment is compatible, so developers don't need to re-write applications for different servers.<br><br>https://www.hpe.com/us/en/what-is/caas.html<br><br><br><br>https://h50146.www5.hpe.com/products/software/oe/linux/mainstream/support/whitepaper/pdfs/4AA6-2761ENW.pdf |
| [1c] the containers of application software excluding a kernel, | In the method practiced by HPE and/or its customer through the Accused Instrumentalities, the containers of application software exclude a kernel.<br><br>*See, e.g.*:<br><br>**Docker container:** A *Docker container* is a lightweight, standalone, executable software package that runs specific services. This software package includes code, runtime, system libraries, configurations, etc. that run as an isolated process in user space. A Docker container is typically used to deploy scalable and repeatable *microservices*. HPE Ezmeral Runtime Enterprise contains innovations around storage, networking, and security to utilize Docker containers as lightweight virtual machines to run Big Data and analytics applications.<br><br>https://support.hpe.com/hpesc/public/docDisplay?docId=a00ecp54hen_us&docLocale=en_US&page=home/about-hpe-ezmeral-container-pl/GEN_Definitions.html |

<table>
<tr><th>Claim 1</th><th>Accused Instrumentalities</th></tr>
<tr><td></td><td>6. Do Docker containers package up the entire OS and make it easier to deploy?<br><br>Docker containers do not package up the OS. They package up the applications with everything that the application needs to run. The engine is installed on top of the OS running on a host. Containers share the OS kernel allowing a single host to run multiple containers.<br><br>https://www.docker.com/blog/the-10-most-common-questions-it-admins-ask-about-docker/<br><br>Containers and VMs perform somewhat similar functions in that they provide virtualized environments in which software applications can run separately from the rest of the system. But these technologies are very different and are used in different situations. Each virtual machine runs both an OS and the application, while containers share a single OS via a kernel, making them more lightweight and portable.<br><br>https://www.hpe.com/us/en/what-is/containers.html</td></tr>
<tr><td>[1d] wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server,</td><td>In the method practiced by HPE and/or its customer through the Accused Instrumentalities, some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server.<br><br>For example, each container will utilize its own local system files, including libraries such as libc/glibc and configuration files, not the corresponding libraries and configuration files of the host OS.<br><br>See, e.g.:<br><br>Docker container: A Docker container is a lightweight, standalone, executable software package that runs specific services. This software package includes code, runtime, system libraries, configurations, etc. that run as an isolated process in user space. A Docker container is typically used to deploy scalable and repeatable microservices. HPE Ezmeral Runtime Enterprise contains innovations around storage, networking, and security to utilize Docker containers as lightweight virtual machines to run Big Data and analytics applications.<br><br>https://support.hpe.com/hpesc/public/docDisplay?docId=a00ecp54hen_us&docLocale=en_US&page=home/about-hpe-ezmeral-container-pl/GEN_Definitions.html<br><br>An application container is a stand-alone, all-in-one package for a software application. Containers include the application binaries, plus the software dependencies and the hardware requirements needed to run, all wrapped up into an independent, self-contained unit.<br><br>https://developer.hpe.com/blog/kubernetes-application-containers-managing-containers-and-cluster-resour/</td></tr>
</table>

| Claim 1 | Accused Instrumentalities |
| --- | --- |
| | ▪ **OS agnostic** – With an application and all its necessary files bundled into one unit – minus an operating system – the container can run on different operating systems, hardware, networks, storage systems and security policies. This means that any environment is compatible, so developers don't need to re-write applications for different servers.<br><br>https://www.hpe.com/us/en/what-is/caas.html |
| [1e] wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server, | In the method practiced by HPE and/or its customer through the Accused Instrumentalities, said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server.<br><br>For example, in some cases the host OS and container will use one or more identical system files, for example when both the host and the container incorporate the same Linux distribution version, or when both host and container use the same version of libc. In other cases modified copies are used instead, for example when different versions of the same library, or configuration files with different parameters, are used by the host and container.<br><br>*See, e.g.*:<br><br>**Docker container:** A *Docker container* is a lightweight, standalone, executable software package that runs specific services. This software package includes code, runtime, system libraries, configurations, etc. that run as an isolated process in user space. A Docker container is typically used to deploy scalable and repeatable *microservices*. HPE Ezmeral Runtime Enterprise contains innovations around storage, networking, and security to utilize Docker containers as lightweight virtual machines to run Big Data and analytics applications.<br><br>https://support.hpe.com/hpesc/public/docDisplay?docId=a00ecp54hen_us&docLocale=en_US&page=home/about-hpe-ezmeral-container-pl/GEN_Definitions.html<br><br>`COPY` and `ADD`: These commands copy files and directories from your local filesystem into the Docker image. They are often used to include your application code, configuration files, and dependencies.<br>https://medium.com/@swalperen3008/what-is-dockerize-and-dockerize-your-project-a-step-by-step-guide-899c48a34df6 |

| Claim 1 | Accused Instrumentalities |
|---|---|
| | ## Container images<br><br>A container image is a ready-to-run software package containing everything needed to run an application: the code and any runtime it requires, application and system libraries, and default values for any essential settings.<br><br>https://kubernetes.io/docs/concepts/containers/ |
| [1f] and wherein the application software cannot be shared between the plurality of secure containers of application software, | In the method practiced by HPE and/or its customer through the Accused Instrumentalities, the application software cannot be shared between the plurality of secure containers of application software.<br><br>For example, each container has an isolated runtime environment that cannot be accessed by other containers, for example including a per-container writeable layer or other ephemeral per-container storage. For another example, when the plurality of secure containers each corresponds to a different container image, each container cannot access another container's image and therefore application software.<br><br>*See, e.g.*:<br><br>**Docker container:** A *Docker container* is a lightweight, standalone, executable software package that runs specific services. This software package includes code, runtime, system libraries, configurations, etc. that run as an isolated process in user space. A Docker container is typically used to deploy scalable and repeatable *microservices*. HPE Ezmeral Runtime Enterprise contains innovations around storage, networking, and security to utilize Docker containers as lightweight virtual machines to run Big Data and analytics applications.<br><br>https://support.hpe.com/hpesc/public/docDisplay?docId=a00ecp54hen_us&docLocale=en_US&page=home/about-hpe-ezmeral-container-pl/GEN_Definitions.html<br><br>Kubernetes namespaces have the following uses:<br>• **Isolation:** Teams, projects, and customers exist in their own environment within a cluster, and do not impact each other's work.<br><br>https://support.hpe.com/hpesc/public/docDisplay?docId=a00ecp55hen_us&docLocale=en_US&page=reference/universal-concepts/Namespaces.html |

| Claim 1 | Accused Instrumentalities |
|---|---|
| | Because each application container creates an isolated environment for its application, the resources allocated to it are the entire machine. Other copies of the same container are "unaware" of each other.<br>https://developer.hpe.com/blog/kubernetes-application-containers-managing-containers-and-cluster-resour/<br># About storage drivers<br>To use storage drivers effectively, it's important to know how Docker builds and stores images, and how these images are used by containers. You can use this information to make informed choices about the best way to persist data from your applications and avoid performance problems along the way.<br>## Storage drivers versus Docker volumes<br>Docker uses storage drivers to store image layers, and to store data in the writable layer of a container. The container's writable layer doesn't persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime. Storage drivers are optimized for space efficiency, but (depending on the storage driver) write speeds are lower than native file system performance, especially for storage drivers that use a copy-on-write filesystem. Write-intensive applications, such as database storage, are impacted by a performance overhead, particularly if pre-existing data exists in the read-only layer.<br>Use Docker volumes for write-intensive data, data that must persist beyond the container's lifespan, and data that must be shared between containers. Refer to the volumes section to learn how to use volumes to persist data and improve performance.<br>https://docs.docker.com/storage/storagedriver/ |

<table>
<tr><th>Claim 1</th><th>Accused Instrumentalities</th></tr>
<tr><td></td><td>

## Images and layers

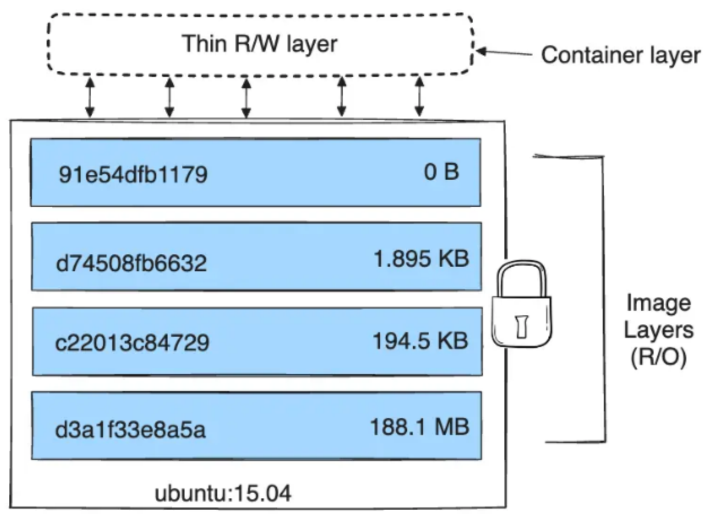
A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only. Consider the following Dockerfile:

```
# syntax=docker/dockerfile:1

FROM ubuntu:22.04
LABEL org.opencontainers.image.authors="org@example.com"
COPY . /app
RUN make /app
RUN rm -r $HOME/.cache
CMD python /app/app.py
```

This Dockerfile contains four commands. Commands that modify the filesystem create a layer. The `FROM` statement starts out by creating a layer from the `ubuntu:22.04` image. The `LABEL` command only modifies the image's metadata, and doesn't produce a new layer. The `COPY` command adds some files from your Docker client's current directory. The first `RUN` command builds your application using the `make` command, and writes the result to a new layer. The second `RUN` command removes a cache directory, and writes the result to a new layer. Finally, the `CMD` instruction specifies what command to run within the container, which only modifies the image's metadata, which doesn't produce an image layer.

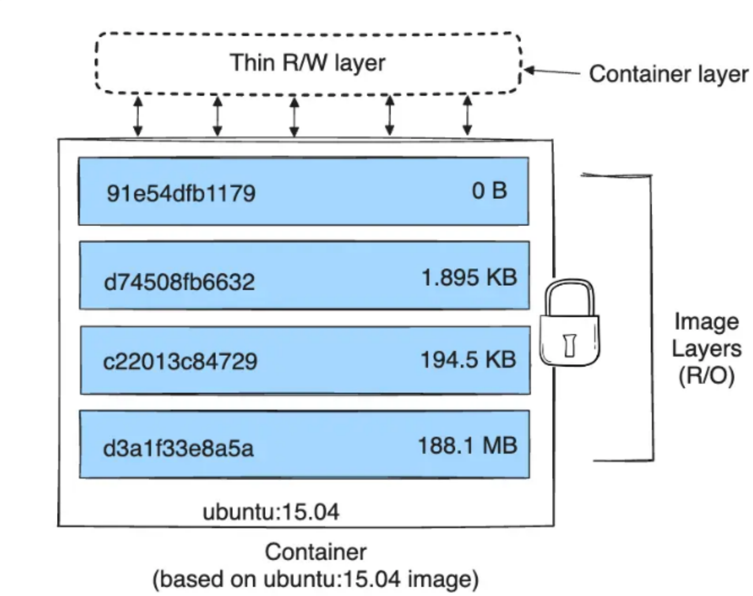
https://docs.docker.com/storage/storagedriver/
</td></tr>
</table>

| Claim 1 | Accused Instrumentalities |
|---|---|
| | Each layer is only a set of differences from the layer before it. Note that both *adding*, and *removing* files will result in a new layer. In the example above, the `$HOME/.cache` directory is removed, but will still be available in the previous layer and add up to the image's total size. Refer to the Best practices for writing Dockerfiles and use multi-stage builds sections to learn how to optimize your Dockerfiles for efficient images.<br><br>The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an `ubuntu:15.04` image.<br><br><br>Container<br>(based on ubuntu:15.04 image)<br><br>https://docs.docker.com/storage/storagedriver/ |

| Claim 1 | Accused Instrumentalities |
|---|---|
| [1g] and wherein each of the containers has a unique root file system that is different from an operating system's root file system. | In the method practiced by HPE and/or its customer through the Accused Instrumentalities, each of the containers has a unique root file system that is different from an operating system's root file system.<br><br>For example, the container's root file system comprises the image layer(s), an ephemeral writeable layer (e.g., in Docker terminology the container layer), and optionally one or more volumes. This root file system is distinct and isolated from the host operating system's root file system.<br><br>*See, e.g.*:<br><br>Using containers isolates software and allows it to work independently across different operating systems, hardware, networks, storage systems, and security policies. It allows the container-based application to transition seamlessly through development, testing, and production environments. Because an operating system is not packed into the container, each container uses minimal computing resources, making it light and easy to install.<br><br>https://www.hpe.com/us/en/what-is/containers.html<br><br>**Node storage:** *Node storage* is storage space available for backing the root file systems of containers. Each HPE Ezmeral Runtime Enterprise host contributes node storage space that is used by the virtual nodes (Docker containers) assigned to that host. The Platform Administrator may optionally specify a quota limiting how much node storage a tenant's virtual nodes may consume.<br><br>https://support.hpe.com/hpesc/public/docDisplay?docId=a00ecp54hen_us&docLocale=en_US&page=home/about-hpe-ezmeral-container-pl/GEN_Definitions.html |

| Claim 1 | Accused Instrumentalities |
|---|---|
| | # About storage drivers<br><br>To use storage drivers effectively, it's important to know how Docker builds and stores images, and how these images are used by containers. You can use this information to make informed choices about the best way to persist data from your applications and avoid performance problems along the way.<br><br>## Storage drivers versus Docker volumes<br><br>Docker uses storage drivers to store image layers, and to store data in the writable layer of a container. The container's writable layer doesn't persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime. Storage drivers are optimized for space efficiency, but (depending on the storage driver) write speeds are lower than native file system performance, especially for storage drivers that use a copy-on-write filesystem. Write-intensive applications, such as database storage, are impacted by a performance overhead, particularly if pre-existing data exists in the read-only layer.<br><br>Use Docker volumes for write-intensive data, data that must persist beyond the container's lifespan, and data that must be shared between containers. Refer to the volumes section to learn how to use volumes to persist data and improve performance.<br><br>https://docs.docker.com/storage/storagedriver/ |

| Claim 1 | Accused Instrumentalities |
| --- | --- |
| | # Images and layers<br><br>A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only. Consider the following Dockerfile:<br><br>`# syntax=docker/dockerfile:1`<br><br>`FROM ubuntu:22.04`<br>`LABEL org.opencontainers.image.authors="org@example.com"`<br>`COPY . /app`<br>`RUN make /app`<br>`RUN rm -r $HOME/.cache`<br>`CMD python /app/app.py`<br><br>This Dockerfile contains four commands. Commands that modify the filesystem create a layer. The `FROM` statement starts out by creating a layer from the `ubuntu:22.04` image. The `LABEL` command only modifies the image's metadata, and doesn't produce a new layer. The `COPY` command adds some files from your Docker client's current directory. The first `RUN` command builds your application using the `make` command, and writes the result to a new layer. The second `RUN` command removes a cache directory, and writes the result to a new layer. Finally, the `CMD` instruction specifies what command to run within the container, which only modifies the image's metadata, which doesn't produce an image layer.<br><br>https://docs.docker.com/storage/storagedriver/ |

| Claim 1 | Accused Instrumentalities |
|---|---|
| | Each layer is only a set of differences from the layer before it. Note that both *adding*, and *removing* files will result in a new layer. In the example above, the `$HOME/.cache` directory is removed, but will still be available in the previous layer and add up to the image's total size. Refer to the Best practices for writing Dockerfiles and use multi-stage builds sections to learn how to optimize your Dockerfiles for efficient images.<br><br>The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an `ubuntu:15.04` image.<br><br><br>https://docs.docker.com/storage/storagedriver/ |

| Claim 1 | Accused Instrumentalities |
| --- | --- |
| | **Volumes**<br><br>Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While bind mounts are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker. Volumes have several advantages over bind mounts:<br><br>https://kubernetes.io/docs/concepts/storage/volumes/<br><br>**Container environment**<br><br>The Kubernetes Container environment provides several important resources to Containers:<br><br>• A filesystem, which is a combination of an image and one or more volumes.<br>• Information about the Container itself.<br>• Information about other objects in the cluster.<br><br>https://kubernetes.io/docs/concepts/containers/container-environment/ |

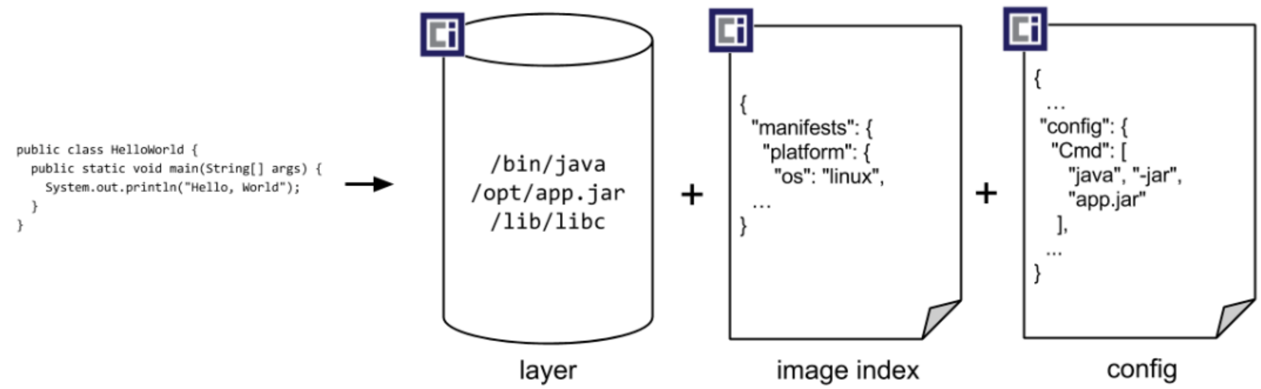
| Claim 1 | Accused Instrumentalities |
| --- | --- |
| | **Images**<br><br>A container image represents binary data that encapsulates an application and all its software dependencies. Container images are executable software bundles that can run standalone and that make very well defined assumptions about their runtime environment.<br><br>You typically create a container image of your application and push it to a registry before referring to it in a Pod.<br><br>https://kubernetes.io/docs/concepts/containers/images/<br><br>**Volumes**<br><br>On-disk files in a container are ephemeral, which presents some problems for non-trivial applications when running in containers. One problem occurs when a container crashes or is stopped. Container state is not saved so all of the files that were created or modified during the lifetime of the container are lost. During a crash, kubelet restarts the container with a clean state. Another problem occurs when multiple containers are running in a `Pod` and need to share files. It can be challenging to setup and access a shared filesystem across all of the containers. The Kubernetes volume abstraction solves both of these problems. Familiarity with Pods is suggested.<br><br>https://kubernetes.io/docs/concepts/storage/volumes/ |

| Claim 1 | Accused Instrumentalities |
|---|---|
| | **Open Container Initiative**<br><br>**Image Format Specification**<br><br>This specification defines an OCI Image, consisting of an image manifest, an image index (optional), a set of filesystem layers, and a configuration.<br><br>The goal of this specification is to enable the creation of interoperable tools for building, transporting, and preparing a container image to run.<br><br>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md |

<table>
<tr><th>Claim 1</th><th>Accused Instrumentalities</th></tr>
<tr><td></td><td>

## Overview

At a high level the image manifest contains metadata about the contents and dependencies of the image including the content-addressable identity of one or more filesystem layer changeset archives that will be unpacked to make up the final runnable filesystem. The image configuration includes information such as application arguments, environments, etc. The image index is a higher-level manifest which points to a list of manifests and descriptors. Typically, these manifests may provide different implementations of the image, possibly varying by platform or other attributes.




https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md
</td></tr>
</table>

<table>
<tr><th>Claim 1</th><th>Accused Instrumentalities</th></tr>
<tr><td></td><td>
<h1>OCI Image Configuration</h1>
An OCI Image is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. This specification outlines the JSON format describing images for use with a container runtime and execution tool and its relationship to filesystem changesets, described in Layers.
This section defines the <code>application/vnd.oci.image.config.v1+json</code> media type.
https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md
</td></tr>
</table>

<table>
<tr><th>Claim 1</th><th>Accused Instrumentalities</th></tr>
<tr><td></td><td>

<h2>Layer</h2>

<ul>
<li>Image filesystems are composed of layers.</li>
<li>Each layer represents a set of filesystem changes in a tar-based layer format, recording files to be added, changed, or deleted relative to its parent layer.</li>
<li>Layers do not have configuration metadata such as environment variables or default arguments - these are properties of the image as a whole rather than any particular layer.</li>
<li>Using a layer-based or union filesystem such as AUFS, or by computing the diff from filesystem snapshots, the filesystem changeset can be used to present a series of image layers as if they were one cohesive filesystem.</li>
</ul>

<h2>Image JSON</h2>

<ul>
<li>Each image has an associated JSON structure which describes some basic information about the image such as date created, author, as well as execution/runtime configuration like its entrypoint, default arguments, networking, and volumes.</li>
<li>The JSON structure also references a cryptographic hash of each layer used by the image, and provides history information for those layers.</li>
<li>This JSON is considered to be immutable, because changing it would change the computed ImageID.</li>
<li>Changing it means creating a new derived image, instead of changing the existing image.</li>
</ul>

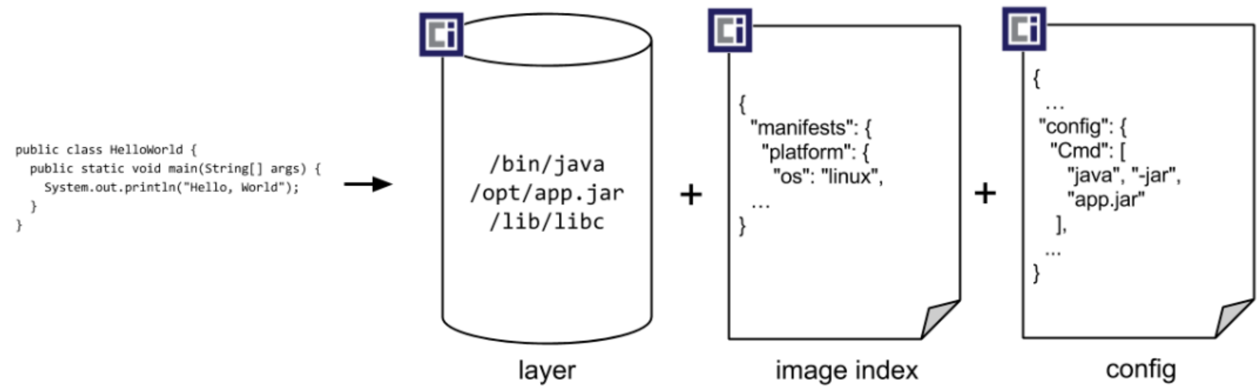
https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md
</td></tr>
</table>

| Claim 1 | Accused Instrumentalities |
| --- | --- |
| | • **rootfs** *object*, REQUIRED<br>The rootfs key references the layer content addresses used by the image. This makes the image config hash depend on the filesystem hash.<br>◦ **type** *string*, REQUIRED<br>MUST be set to `layers`. Implementations MUST generate an error if they encounter a unknown value while verifying or unpacking an image.<br>◦ **diff_ids** *array of strings*, REQUIRED<br>An array of layer content hashes (`DiffIDs`), in order from first to last.<br>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md |

**Claim 2**

| Claim 2 | Accused Instrumentalities |
| --- | --- |
| 2. A method as defined in claim 1, wherein each container has an execution file associated therewith for starting the one or more applications. | HPE and/or its customer practices, through the Accused Instrumentalities, a method as defined in claim 1, wherein each container has an execution file associated therewith for starting the one or more applications.<br>For example, a container image has an associated image configuration comprising information for starting the one or more applications. This can be an Open Containers Initiative image configuration.<br>*See, e.g.*: |

| Claim 2 | Accused Instrumentalities |
| --- | --- |
| | # Open Container Initiative<br>## Image Format Specification<br>This specification defines an OCI Image, consisting of an image manifest, an image index (optional), a set of filesystem layers, and a configuration.<br>The goal of this specification is to enable the creation of interoperable tools for building, transporting, and preparing a container image to run.<br>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md |

<table>
<tr><th>Claim 2</th><th>Accused Instrumentalities</th></tr>
<tr><td></td><td>
<h2>Overview</h2>
At a high level the image manifest contains metadata about the contents and dependencies of the image including the content-addressable identity of one or more filesystem layer changeset archives that will be unpacked to make up the final runnable filesystem. The image configuration includes information such as application arguments, environments, etc. The image index is a higher-level manifest which points to a list of manifests and descriptors. Typically, these manifests may provide different implementations of the image, possibly varying by platform or other attributes.


https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md
</td></tr>
</table>

| Claim 2 | Accused Instrumentalities |
|---|---|
| | # OCI Image Configuration<br><br>An OCI *Image* is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. This specification outlines the JSON format describing images for use with a container runtime and execution tool and its relationship to filesystem changesets, described in Layers.<br><br>This section defines the `application/vnd.oci.image.config.v1+json` media type.<br><br>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md |

| Claim 2 | Accused Instrumentalities |
| --- | --- |
| | • **config** *object*, OPTIONAL<br><br>The execution parameters which SHOULD be used as a base when running a container using the image. This field can be `null`, in which case any execution parameters should be specified at creation of the container.<br><br>○ **Env** *array of strings*, OPTIONAL<br><br>Entries are in the format of `VARNAME=VARVALUE`. These values act as defaults and are merged with any specified when creating a container.<br><br>○ **Entrypoint** *array of strings*, OPTIONAL<br><br>A list of arguments to use as the command to execute when the container starts. These values act as defaults and may be replaced by an entrypoint specified when creating a container.<br><br>○ **Cmd** *array of strings*, OPTIONAL<br><br>Default arguments to the entrypoint of the container. These values act as defaults and may be replaced by any specified when creating a container. If an `Entrypoint` value is not specified, then the first entry of the `Cmd` array SHOULD be interpreted as the executable to run.<br><br>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md |

**Claim 6**

| Claim 6 | Accused Instrumentalities |
|---|---|
| 6. A method as defined in claim 2, comprising the step of assigning a unique associated identity to each of a plurality of the containers, wherein the identity includes at least one of IP address, host name, and MAC address. | HPE and/or its customer practices, through the Accused Instrumentalities, a method as defined in claim 2, comprising the step of assigning a unique associated identity to each of a plurality of the containers, wherein the identity includes at least one of IP address, host name, and MAC address.<br><br>For example, Kubernetes containers have an associated hostname, which in the case of a single-container Pod is the unique identity of that container. For another example, Kubernetes pods have an associated hostname, which is unique. For another example, a networked Kubernetes pod has an assigned IPv4 and/or IPv6 address. For another example, a Docker container has an IP address and a hostname.<br><br>*See, e.g.*:<br><br>**Container information**<br><br>The *hostname* of a Container is the name of the Pod in which the Container is running. It is available through the `hostname` command or the `gethostname` function call in libc.<br><br>The Pod name and namespace are available as environment variables through the downward API.<br><br>User defined environment variables from the Pod definition are also available to the Container, as are any environment variables specified statically in the container image.<br><br>https://kubernetes.io/docs/concepts/containers/container-environment/ |

| Claim 6 | Accused Instrumentalities |
|---|---|
| | **IP address and hostname**<br><br>By default, the container gets an IP address for every Docker network it attaches to. A container receives an IP address out of the IP subnet of the network. The Docker daemon performs dynamic subnetting and IP address allocation for containers. Each network also has a default subnet mask and gateway.<br><br>You can connect a running container to multiple networks, either by passing the `--network` flag multiple times when creating the container, or using the `docker network connect` command for already running containers. In both cases, you can use the `--ip` or `--ip6` flags to specify the container's IP address on that particular network.<br><br>In the same way, a container's hostname defaults to be the container's ID in Docker. You can override the hostname using `--hostname`. When connecting to an existing network using `docker network connect`, you can use the `--alias` flag to specify an additional network alias for the container on that network.<br><br>https://docs.docker.com/network/ |

**Claim 9**

| Claim 9 | Accused Instrumentalities |
|---|---|
| 9. A method as defined in claim 2, wherein server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation is associated with at least some of the containers prior to the applications within the containers being executed. | HPE and/or its customer practices, through the Accused Instrumentalities, a method as defined in claim 2, wherein server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation is associated with at least some of the containers prior to the applications within the containers being executed.<br><br>For example, Kubernetes tracks and limits resource usage, including CPU and memory resources. For another example, Docker tracks and limits resource usage, including CPU and memory resources.<br><br>*See, e.g.*: |

| | |
|---|---|
| | **Resource Management for Pods and Containers**<br><br>When you specify a Pod, you can optionally specify how much of each resource a container needs. The most common resources to specify are CPU and memory (RAM); there are others.<br><br>When you specify the resource *request* for containers in a Pod, the kube-scheduler uses this information to decide which node to place the Pod on. When you specify a resource *limit* for a container, the kubelet enforces those limits so that the running container is not allowed to use more of that resource than the limit you set. The kubelet also reserves at least the *request* amount of that system resource specifically for that container to use.<br><br>Requests and limits<br>If the node where a Pod is running has enough of a resource available, it's possible (and allowed) for a container to use more resource than its `request` for that resource specifies. However, a container is not allowed to use more than its resource `limit`.<br><br>For example, if you set a `memory` request of 256 MiB for a container, and that container is in a Pod scheduled to a Node with 8GiB of memory and no other Pods, then the container can try to use more RAM.<br><br>If you set a `memory` limit of 4GiB for that container, the kubelet (and container runtime) enforce the limit. The runtime prevents the container from using more than the configured resource limit. For example: when a process in the container tries to consume more than the allowed amount of memory, the system kernel terminates the process that attempted the allocation, with an out of memory (OOM) error. |

| Claim 9 | Accused Instrumentalities |
| --- | --- |
| | Limits can be implemented either reactively (the system intervenes once it sees a violation) or by enforcement (the system prevents the container from ever exceeding the limit). Different runtimes can have different ways to implement the same restrictions.<br><br>https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/<br><br>**Runtime options with Memory, CPUs, and GPUs**<br>By default, a container has no resource constraints and can use as much of a given resource as the host's kernel scheduler allows. Docker provides ways to control how much memory, or CPU a container can use, setting runtime configuration flags of the `docker run` command. This section provides details on when you should set such limits and the possible implications of setting them.<br><br>**Limit a container's access to memory**<br>Docker can enforce hard or soft memory limits.<br>• Hard limits lets the container use no more than a fixed amount of memory.<br>• Soft limits lets the container use as much memory as it needs unless certain conditions are met, such as when the kernel detects low memory or contention on the host machine.<br><br>https://docs.docker.com/config/containers/resource_constraints/ |

**Claim 10**

| Claim 10 | Accused Instrumentalities |
| --- | --- |
| 10. A method as defined in claim 2, wherein in operation when an application residing within a container is executed, said application has no access to system files or applications in other containers or | HPE and/or its customer practices, through the Accused Instrumentalities, a method as defined in claim 2, wherein in operation when an application residing within a container is executed, said application has no access to system files or applications in other containers or to system files within the operating system during execution thereof. |

| Claim 10 | Accused Instrumentalities |
|---|---|
| to system files within the operating system during execution thereof. | *See, e.g.*:<br><br>**Docker container:** A *Docker container* is a lightweight, standalone, executable software package that runs specific services. This software package includes code, runtime, system libraries, configurations, etc. that run as an isolated process in user space. A Docker container is typically used to deploy scalable and repeatable *microservices*. HPE Ezmeral Runtime Enterprise contains innovations around storage, networking, and security to utilize Docker containers as lightweight virtual machines to run Big Data and analytics applications.<br><br>https://support.hpe.com/hpesc/public/docDisplay?docId=a00ecp54hen_us&docLocale=en_US&page=home/about-hpe-ezmeral-container-pl/GEN_Definitions.html<br><br>Kubernetes namespaces have the following uses:<br>• **Isolation:** Teams, projects, and customers exist in their own environment within a cluster, and do not impact each other's work.<br><br>https://support.hpe.com/hpesc/public/docDisplay?docId=a00ecp55hen_us&docLocale=en_US&page=reference/universal-concepts/Namespaces.html<br><br>Because each application container creates an isolated environment for its application, the resources allocated to it are the entire machine. Other copies of the same container are "unaware" of each other.<br>https://developer.hpe.com/blog/kubernetes-application-containers-managing-containers-and-cluster-resour/ |

**Claim 31**

| Claim 31 | Accused Instrumentalities |
|---|---|
| [31pre] A computing system for performing a plurality of tasks each comprising a plurality of processes comprising: | To the extent the preamble is construed as a limitation, each Accused Instrumentality is or comprises a computing system for performing a plurality of tasks each comprising a plurality of processes.<br><br>*See* claim limitations below. *See also* analysis and evidence for [1pre] above. |
| [31a] a system having a plurality of secure containers of associated files accessible to, and for execution on, one or more servers, each container being mutually exclusive of the other, such that read/write files within a container cannot | Each Accused Instrumentality comprises a system having a plurality of secure containers of associated files accessible to, and for execution on, one or more servers, each container being mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers, each container of files is said to have its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a Mac_address. |

| Claim 31 | Accused Instrumentalities |
|---|---|
| be shared with other containers, each container of files is said to have its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a Mac_address | *See* analysis and evidence for [1pre], limitations [1a] and [1f], and claim 6 above. |
| [31b] wherein, the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes, and associated system files for use in executing the one or more processes wherein the associated system files are files that are copies of files or modified copies of files that remain as part of the operating system, each container having its own execution file associated therewith for starting one or more applications, in operation, each container utilizing a kernel resident on the server and wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel; and, | Each Accused Instrumentality comprises a system wherein the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes, and associated system files for use in executing the one or more processes wherein the associated system files are files that are copies of files or modified copies of files that remain as part of the operating system, each container having its own execution file associated therewith for starting one or more applications, in operation, each container utilizing a kernel resident on the server and wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel.<br><br>*See* analysis and evidence for [1pre], limitations [1a], [1c], [1d], [1e], and [1f], and claim 2 above. |
| [31c] a run time module for monitoring system calls from applications associated with one or more containers and for providing control of the one or more applications. | Each Accused Instrumentality comprises a run time module for monitoring system calls from applications associated with one or more containers and for providing control of the one or more applications.<br><br>For example, HPE Ezmeral Runtime Enterprise includes either the Docker or containerd runtime module. For another example, Kubernetes uses the Linux kernel's seccomp mode to monitor and control system calls made from a container.<br><br>*See, e.g.*: |

| Claim 31 | Accused Instrumentalities |
|---|---|
| | **Runtime is `containerd`**<br><br>The Kubernetes distribution provided with HPE Ezmeral Runtime Enterprise is based on the `containerd` runtime.<br><br>The `containerd` runtime is used on all hosts except for the following:<br><br>• The HPE Ezmeral Runtime Enterprise control plane hosts (Controller, Shadow Controller, Arbiter, and Gateway), which continue to use the Docker runtime.<br>• Kubernetes clusters that were created in on deployments running releases prior to HPE Ezmeral Runtime Enterprise 5.5.0 that are now on a deployment that has been upgraded to HPE Ezmeral Runtime Enterprise 5.5.0 or later. These legacy clusters are supported for a limited time. See Kubernetes Cluster Types and Compatibility.<br><br>https://docs.ezmeral.hpe.com/runtime-enterprise/56/reference/kubernetes/hewlett_packard_enterprise_distributions_of_kubernetes.html |

| Claim 31 | Accused Instrumentalities |
|---|---|
| | **Container Runtimes**<br><br>**Note:** Dockershim has been removed from the Kubernetes project as of release 1.24. Read the Dockershim Removal FAQ for further details.<br><br>You need to install a container runtime into each node in the cluster so that Pods can run there. This page outlines what is involved and describes related tasks for setting up nodes.<br><br>Kubernetes 1.30 requires that you use a runtime that conforms with the Container Runtime Interface (CRI).<br><br>See CRI version support for more information.<br>https://kubernetes.io/docs/setup/production-environment/container-runtimes/ |

| Claim 31 | Accused Instrumentalities |
|---|---|
| | **Restrict a Container's Syscalls with seccomp**<br><br>ⓘ **FEATURE STATE:** `Kubernetes v1.19 [stable]`<br><br>Seccomp stands for secure computing mode and has been a feature of the Linux kernel since version 2.6.12. It can be used to sandbox the privileges of a process, restricting the calls it is able to make from userspace into the kernel. Kubernetes lets you automatically apply seccomp profiles loaded onto a node to your Pods and containers.<br><br>Identifying the privileges required for your workloads can be difficult. In this tutorial, you will go through how to load seccomp profiles into a local Kubernetes cluster, how to apply them to a Pod, and how you can begin to craft profiles that give only the necessary privileges to your container processes.<br><br>https://kubernetes.io/docs/tutorials/security/seccomp/ |